USB I/O Programming Manual

Delcom Engineering
200 William Street
Port Chester, NY 10573
914-934-5170
914-934-5171 Fax
www.delcom-eng.com

Document Ver 1.6 October 08, 2002

Contents

1.0 Introduction

This document describes how to communicate to the Delcom USB I/O device. Please use this document along with the sample code available on the web site.

2.0 Requirements

Delcom USB IO Device
Delcom IO USB Driver
USB ready computer running Win98, ME or Win 2000
Microsoft Visual C++ version 4.0 or better, or equivalent 32 bit compiler for x86 based system, or Visual basic
Knowledge of C or Visual basic

3.0 Overview

To communicate with USB IO Driver one must first enumerate the device. The enumeration of the device returns a device name. This device name is used to open the interface, using CreateFile(). Once you have the handle from CreateFile() you can use DeviceIOControl() to communicate to the USB IO Device and CloseHandle() to close it. The hardest part is getting the device name the rest is simply. To send commands to the USB IO device simply build a command packet and submit it using the DeviceIOControl functions.

4.0 Device Enumeration:

In order to communicate to the USB device one must first find its device name. The device name consists of a number representing a physical port plus the GUID (global unique identifier) for the device. The current USB port and the GUID are combined to form the device name. The device name can change each time you plug in an additional device or plug the device into a different USB port or hub on your computer.

The GUID for the Delcom USB I/O device is {b5157d69-75f8-11d3-8ce0-00207815e611}, and a typical complete device name looks like \\.\0000000000000012#{b5157d69-75f8-11d3-8ce0-00207815e611}.

In Windows XP a typical device name look like \\.\USB#Vid_0FC5&Pid_1222#000000001#{b5157d69-75f8-11d3-8ce0-00207815e611}.

4.1 Device Name Registry Method

There are two ways to get the device name. The easiest method is to read the device name from the registry. When a USB I/O device is plugged in to your computer, the OS detects it and loads the driver. When this driver loads the device name is stored in the registry. Then the user just reads the name out of the registry. This method has one disadvantage. It can't be used when more than one USB I/O device is plugged in to your computer, because only the last device name will be recorded in the registry.

To use the registry method simply open the registry key, query the value, and close the registry key. The registry key name is `DeviceName` and the path is
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Delcom\USBIODS\Parameters\

You can uses regedit.exe to find the entry. It also a good place to copy the GUID from so you don't any mistakes.

4.2 VB Registry Example

Here is an example in Visual Basic on how to read the device name from the registry.

```
DIM DeviceName as string
DeviceName = GetRegValue(HKEY_LOCAL_MACHINE, _
            "System\CurrentControlSet\Services\Delcom\USBIODS\Parameters\", _
            "DeviceName")


' GetRegValue - Gets the Key value in the registry given a registry key.
Function GetRegValue(hKey As Long, lpszSubKey As String, szKey As String ) As
String

Dim phkResult As Long, lResult As Long
Dim szBuffer As String, lBuffSize As Long

'Create Buffer
szBuffer = Space(255)           ' Allocate buffer space
lBuffSize = Len(szBuffer)       ' Set the length

RegOpenKeyEx hKey, lpszSubKey, 0, 1, phkResult      'Open the Key, get a
handle to it

lResult = RegQueryValueEx(phkResult, szKey, 0, 0, szBuffer, lBuffSize) 'Query
the value

RegCloseKey phkResult           'Close the Key

If lResult = ERROR_SUCCESS Then
    GetRegValue = szBuffer           ‘ return key value
End If
Exit Function
```

4.3 Device Name Enumeration Method

The second method to get the device name is two use Windows™ device manger. To do this one calls a function in the setupapi.dll. Simply poll the device manger with the USB I/O GUID for all the devices that match the GUID given. The device manger will return the device names for all the devices currently available on your system. This is the better way of getting the device name. It allows the user to use multiple devices on the same computer. The draw back is that it is little more complicated.

4.4 C Enumeration Example

Below is a C example using this enumeration method.

Use the DEFINE_GUID macro to build the GUID.

```
// {B5157D69-75F8-11d3-8CE0-00207815E611}
DEFINE_GUID(USBIODS_GUID,
0xb5157d69, 0x75f8, 0x11d3, 0x8c, 0xe0, 0x0, 0x20, 0x78, 0x15,
0xe6, 0x11);
```

This GUID is passed to SetupDiGetClassDevs(), which returns a handle to the device. The enumeration functions are found in the setupapi library.

```
HDEVINFO hinfo = SetupDiGetClassDevs(&USBIODS_GUID, NULL,
NULL, DIGCF_PRESENT | DIGCF_INTERFACEDEVICE);
```

The first argument identifies the interface you're looking for. The flag bits in the last argument indicate that you are looking for the interfaces exported by the USB I/O device.

Once you have a handle to the device information set, you can perform an enumeration of all the devices that export the particular interface you're interested in. See Microsoft function documentation for more information on setupapi.dll library functions.

Poll the device manager till there are no matching devices left.

```
   int i;
   Cstring Devices[10];              // an array of cstrings
   for (DWORD i=0; ; ++i)
      {
      SP_INTERFACE_DEVICE_DATA Interface_Info;
      Interface_Info.cbSize = sizeof(Interface_Info);
      // Enumerate device
      if (!SetupDiEnumInterfaceDevice(hInfo, NULL, (LPGUID)
            &USBIODS_GUID,i, &Interface_Info))
         {
         SetupDiDestroyDeviceInfoList(hInfo);
         return(i);
         }

      DWORD needed;                 // get the required lenght
      SetupDiGetInterfaceDeviceDetail(hInfo, &Interface_Info,
            NULL, 0, &needed, NULL);
PSP_INTERFACE_DEVICE_DETAIL_DATA detail =
(PSP_INTERFACE_DEVICE_DETAIL_DATA) malloc(needed);
      if (!detail)
```

```
         {
         SetupDiDestroyDeviceInfoList(hInfo);
         return(i);
         }
                                    // fill the device details
      detail->cbSize =
            sizeof(SP_INTERFACE_DEVICE_DETAIL_DATA);
      if (!SetupDiGetInterfaceDeviceDetail(hInfo,
         &Interface_Info,detail, needed,NULL, NULL))
         {
         free((PVOID) detail);
         SetupDiDestroyDeviceInfoList(hInfo);
         return(i);
         }

      char name[MAX_PATH];
      strncpy(name, detail->DevicePath, sizeof(name));
      free((PVOID) detail);
Devices[i] = name;      // keep a copy of each device name
      }  // end of for loop
```

After this code runs you end up with a list of device names, or NULL if no devices could be found (i = 0). Each device name will represent one USB I/O device that is plugged into your computer. If you know that you will only support one USB I/O device on your system at one time, you can reduce the enumeration code by dropping the for loop and only going through the code once. The device name(s) that are returned from the above code have a port number prefixed to the original GUID. The port number is related to order of the plug and play devices on your machine and can not be predetermined. The device name should look like the following.

```
\\.\000000000000012#{b5157d69-75f8-11d3-8ce0-00207815e611}
```

This is the complete device name one will use in order to communicate with the USB I/O device.

5.0 Device Communications:

5.1 Open Device

To begin communicating with the USB I/O device you must first acquire a handle to it. To do this just pass the device name to the CreateFile() function. This is done in the same manner as opening or creating a file. If successful, this function will return a handle to the device. If the device is not plugged in, un-powered, or opened by another program this function will fail.

```
HANDLE hUsbDevice = CreateFile(devicename, GENERIC_READ |
GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
```

5.2 Device Close

When your application has finished using the device, the device should be closed. To do this call CloseHandle() with the device handle. If you do not close the device, you will not be able to access it again without re-setting the device port.

```
CloseHandle( hUsbDevice ) ;
```

6.0 Device Communications

6.1 Device I/O Control

The DeviceIOControl() function provides the communication method between the users and the device. This function accepts CTL_CODES and users buffers that are passed to the device driver and eventually the USB device.

```
success = DeviceIoControl(hUsbDevice,
        IOCTL_USBIODS_SEND_PACKET,
        &TxPacket, 8+TxPacket->Length,
        &RxPacket, 8,
        &nBytes, NULL);
```

The CTL Codes are predefined codes that describe the action to take place. There are many different codes but for our purposes we are only concerned with the send packet code.

Below is the CTL_CODE generation shown in C.

```
#define CTL_CODE( DeviceType, Function, Method, Access ) ( \
    ((DeviceType) << 16) | ((Access) << 14) | ((Function) << 2) | (Method))

#define METHOD_BUFFERED                 0
#define FILE_ANY_ACCESS                 0
#define FILE_DEVICE_UNKNOWN             0x00000022

// ------------------------------------------------------- //
#define DELCOM_USBIO_IOCTL_VENDOR 0x0800          // Vendor defined

#define IOCTL_USBIO_WRITE_PACKET     CTL_CODE(FILE_DEVICE_UNKNOWN,  \
                                     DELCOM_USBIO_IOCTL_VENDOR+10,\
                                     METHOD_BUFFERED,  \
                                     FILE_ANY_ACCESS)
```

The above code generates a CTL_CODE of 0x222028. You can just use this number instead for using the above code, see below. For VB code use &H222028.

```
#define IOCTL_USBIO_SEND_PACKET 0x222028    // for C
Const CTL_CODE_SEND_PACKET = &H222028       '  for VB
```

Once you have the CTL_CODE the next step is to make the command packet. This is a simply structure of which you just set the fields for a particular command. The fields in the command packet are described in the USB IO Data Sheet. Simply fill the structure and send it to the USB device with the DeviceIOControl function. For read commands the DeviceIOControl function returns the data in the RxPacket. The length of the sent packet can range from 8 to 16 bytes. The received packet is always 8 bytes long. The length data member of the send packet is the length of the data extension. The data extension is only required by certain commands. If the data extension member is not being used set the data extension length to zero.

The packet command structure consists of the following elements.

```
// Command Packet Structure define in C
typedef struct _ioPacket{
      unsigned char  Recipient;
      unsigned char  DeviceModel;
      unsigned char MajorCmd;
      unsigned char MinorCmd;
      unsigned char DataLSB;
      unsigned char DataMSB;
      unsigned short Length; // length of data extension
      unsigned char DataExtension[8];
} VENDORPACKET,*PVENDORPACKET;

‘ Command Packet Structure define in VB
Public Type PacketStructure
    Recipient   As Byte
    DeviceModel As Byte
    MajorCmd    As Byte
    MinorCmd    As Byte
    DataLSB     As Byte
    DataMSB     As Byte
    Length      As Int      ‘ length of data extension
    DataEntension(8)   As Byte

End Type
```

6.2 C Example

This C code example sends the packet and receives the data in the same packet that was sent to it. On error it returns –1.

```
int UsbIoCtrl(PVENDORPACKET pPacket)
      {
      ULONG nBytes;
      BOOLEAN Success;

      Success = DeviceIoControl(hUsb,
            IOCTL_USBIO_WRITE_PACKET,
            pPacket, 8+pPacket->Length, pPacket, 8, &nBytes,
            NULL) ;

      if(!Success ) //|| ( nBytes != sizeof(VENDORPACKET) ) )
            {
            if(Verbose)MessagePopup ("UsbIoCtrl
                  Error","DeviceIoControl call failed!");
            return(-1);
            }
      else
            return(0);
      }
```

6.3 VB Example

This VB code example sends the packet and receives the data in the returned value of the function.

```
'Sends the USB packet to the device
Function SendPacket(ByRef TxPacket As PacketStructure) As PacketStructure
Dim lpResult As Long
Dim RxPacket As PacketStructure

TxPacket.Recipient = 8      ' always 8
TxPacket.DeviceModel = 18   ' always 18

    If 0 = DeviceIoControl(hDevice, CTL_CODE_SEND_PACKET, TxPacket,
        8+TxPacket.Length, RxPacket, 8, lpResult, 0) Then
            MsgBox "SendPacket() DeviceIoControl Failed."
            Exit Function
    End If

SendPacket = RxPacket

End Function
```

7.0 Registry Keys:

The following is a list of registry keys that the USB I/O driver adds to the registry. To access the registry, run RegEdit.exe from the command prompt. The registry keys are located at:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Delcom\USBIODS\Parameters
```

Delcom USB I/O Registry Keys

- DebugLevel — Used for debugging should always be zero.
- BootUpTest — Used for testing should always be zero.
- DeviceName — This string contains the device name of the last USB IO device loaded.

Appendix A: Related Documents and Web sites

Universal Serial Bus Specification → www.usb.org
Microsoft Development Network → www.msdn.microsoft.com
Delcom Engineering Web Site → www.delcom-eng.com